The AMC Scheduling Problem: A Description for Reproducibility

Laurence A. Kramer and **Stephen F. Smith**
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{lkramer,sfs}@cs.cmu.edu

Abstract

The United States Air Force Air Mobility Command (AMC) is responsible for managing hundreds of airlift and air refueling missions every week on a global scale. Allocating airframes and flight crews to individual missions is a large and complex problem. We have built the AMC Airlift and Air Refueling Allocator to assist planners and schedulers in this task, and the tool is currently transitioning into daily operations. Development of this system has involved us in a number of fruitful areas of research, ranging from building a model that allows end users to interact with the system at varying levels of automation to exploring techniques for maximizing the number of missions assigned in an environment of resource scarcity. In this technical report we present a description of the AMC scheduling problem in a somewhat abstract form, so that interested parties may experiment with it. We provide specifications for building the model, provide pointers to input data, and provide end results for scheduling missions while varying capacity constrainedness.

1 Introduction

Over the past several years, we have studied a task swapping algorithm, *TaskSwap*, [Kramer and Smith, 2003; 2004a; 2004b; 2005] as a means of repairing oversubscribed schedules by judicious task retraction and reinsertion. The setting for this work is the AMC domain[Becker and Smith, 2000; Smith *et al.*, 2004], a large, multi-capacity, multi-resource mission scheduling problem with fixed time windows and resource-dependent task durations. In this paper we provide a somewhat simplified specification of the AMC domain so that interested researchers may reproduce and extend our techniques, and compare them with alternate methods for scheduling large numbers of missions in the face of restricted resource capacity.

We have "simplified" the actual problem so that the level of effort required to re-implement the domain model is not too onerous, and at the same time we have tried to maintain those elements of the problem that make it interesting and challenging. There is always the danger in producing a "benchmark" problem set that the problems become so abstract that they bear no resemblance to the real world domains that inspired them. We feel fairly confident that our abstractions balance a need for ease of implementation while retaining the critical characteristics of the actual domain.

In the remainder of the paper we first summarize the AMC problem domain, then go on to specify in detail a means to implementing an executable model, including specifications for some key algorithms. As we proceed, we point out where the simplified domain model differs from the actual deployed model, mainly to inform others as to some of the complexities of "actual" problem. In pointing out these differences, we don't attempt to document the specification of the real world model in full detail.

While the interested reader should be able to implement a *TaskSwap* procedure based on the descriptions in the documents cited above, we **do not** provide a specification for a general purpose scheduling engine such as the one we employ. We assume that such an engine is already available to those with the interest to reproduce our work.

Finally, we report results of experiments with the *TaskSwap* as applied to the abstract AMC model with supplied data sets.

2 The AMC Scheduling Problem

Here we reproduce a concise description of the problem taken from [Kramer and Smith, 2005]: "The AMC scheduling problem can be characterized abstractly as follows:

- A set T of tasks (or missions) are submitted for execution. Each task $i \in T$ has an earliest pickup time est_i, a latest delivery time lft_t, a pickup location $orig_i$, a dropoff location $dest_i$, a duration d_i (determined by $orig_i$ and $dest_i$) and a priority pr_i

- A set Res of resources (or air wings) are available for assignment to missions. Each resource $r \in Res$ has capacity $cap_r \geq 1$ (corresponding to the number of contracted aircraft for that wing).

- Each task i has an associated set Res_i of feasible resources (or air wings), any of which can be assigned to carry out i. Any given task i requires 1 unit of capacity (i.e., one aircraft) of the resource r that is assigned to perform it.

1. REPORT DATE **NOV 2005**	2. REPORT TYPE	3. DATES COVERED **00-00-2005 to 00-00-2005**
4. TITLE AND SUBTITLE **The AMC Scheduling Problem: A Description for Reproducibility**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Robotics Institute,Pittsurgh,PA,15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
tech. report CMU-RI-TR-05-75

14. ABSTRACT

The United States Air Force Air Mobility Command (AMC) is responsible for managing hundreds of airlift and air refueling missions every week on a global scale. Allocating airframes and flight crews to individual missions is a large and complex problem. We have built the AMC Airlift and Air Refueling Allocator to assist planners and schedulers in this task, and the tool is currently transitioning into daily operations. Development of this system has involved us in a number of fruitful areas of research ranging from building a model that allows end users to interact with the system at varying levels of automation to exploring techniques for maximizing the number of missions assigned in an environment of resource scarcity. In this technical report we present a description of the AMC scheduling problem in a somewhat abstract form, so that interested parties may experiment with it. We provide specifications for building the model, provide pointers to input data, and provide end results for scheduling missions while varying capacity constrainedness.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

- Each resource r has a designated location $home_r$. For a given task i, each resource $r \in Res_i$ requires a positioning time $pos_{r,i}$ to travel from $home_r$ to $orig_i$, and a de-positioning time $depos_{r,i}$ to travel from $dest_i$ back to $home_r$.

A schedule is a *feasible* assignment of missions to wings. To be feasible, each task i must be scheduled to execute within its $[est_i, lft_i]$ interval, and for each resource r and time point t, $assigned\text{-}cap_{r,t} \leq cap_r$. Typically, the problem is over-subscribed and only a subset of tasks in T can be feasibly accommodated. If all tasks cannot be scheduled, preference is given to higher priority tasks. Tasks that cannot be placed in the schedule are designated as *unassignable*.

3 Building the "Abstract" AMC Model

Building an "abstract" AMC model is comprised of generating *missions* with an itinerary of *geographical locations* and *air wings* that exist at a specific location and which possess a given number of *aircraft* which can be assigned to a mission. Data for these objects are found in four tab-delimited text files that we provide: port-data.txt for locations, wing-data.txt for air wings, mds-data.txt[1] for aircraft, and mission-data.txt for missions.

3.1 Building Missions

In the abstract model, missions are defined as an ordered set of legs, each leg being the movement of an aircraft from point A to point B. An itinerary of legs is concisely represented in the mission data file by a list of locations. For instance, for mission 6UN45P901337 the itinerary is "KWRI KIWI KWRI." This requires that the aircraft fly from location KWRI to KIWI and then from KIWI back to KWRI. More precisely, a mission is delineated by seven fields in the file mission-data.txt:

- **Mission ID.** This is an alphanumeric string, possibly with embedded blanks, which uniquely designates the mission.

- **Priority.** This is an alphanumeric string of length three, whose first character is a non-zero integer, second character a letter, and third character a non-zero integer. "Lower" values denote a higher priority mission, so for instance, 1A1 is the highest priority, and 1A3 is a higher priority than 1B1. When missions are assigned, higher priority missions must be given preference for assignment if at all possible. That is, scheduling two lower priority missions is not preferred over one higher priority mission; however if a high priority mission cannot be feasibly inserted into a schedule, it is permissable to assign a lower priority one.

- **Aircraft.** This field denotes the aircraft type that must be used to fly the mission. In the actual application it is sometimes possible to substitute a different aircraft type, but for the purposes of the abstract model, we'll assume a unique aircraft type per mission. We also make the

simplifying assumption that each mission requires exactly one aircraft. In the full application one or more air crews are also assigned to missions; however for our purposes here, crew assignment will be ignored.

- **Release (Date).** This field, in MM-DD-YYYY-HH:MM format, represents the time on or after which the required part (non-positioning legs) of the mission itinerary *must* begin. I.e., this represents the beginning of the feasible window for the mission. The first non-positioning leg may begin at any time on or after this time as long as all (non-depositioning) legs can be executed to completion on or before the Due Date. The significance of positioning and depositioning legs will be described shortly.

- **Touchdown (Date).** For most missions, this field, in MM-DD-YYYY-HH:MM format, is exactly the same as the Due Date (see below), however some missions are specified with a simple one-leg itinerary where the first and last stops are the same, for instance "KIAD KIAD." What this indicates is that the assigned aircraft will remain at KIAD for a (fixed) period of time which can be computed as the difference between the Touchdown Date and the Release Date.[2]

- **Due (Date).** This field, in MM-DD-YYYY-HH:MM format, represents the time on or before which the required part (non-depositioning legs) of the mission itinerary *must* end. I.e., this represents the end of the feasible window for the mission. The last non-depositioning leg may end at any time on or before this time as long as all (non-positioning and non-depositioning) legs can be executed to completion on or after the Release Date and on or before the Due Date.

- **Itinerary.** As mentioned earlier, the itinerary is a sequence of alphanumeric designations for locations or stops which an aircraft assigned to a mission must visit. Given the input itinerary "RJTY RKSO RJTY PAED," for instance, a good way to think about this (which closely approximates what actually takes place) is that an aircraft assigned to that mission picks up some cargo at RJTY and flies to RKSO where it unloads that cargo and loads some new cargo. After this it flies back to RJTY, unloads some of the cargo, and then flies to PAED, where it unloads the rest of the cargo. When we construct the mission, the important thing is that we think of the specified itinerary as "cargo-carrying legs." For the abstract model we will not consider any of the support activities that are actually managed in the full model such as times to refuel, load, and unload the aircraft, and mandatory time periods for the crew to rest.

Given the cargo-carrying legs in the input data for a mission, there are two additional legs that are left unspecified, the positioning leg and the depositioning leg. Note that for every mission that is to be assigned an aircraft, the aircraft must fly from some location to the first cargo-carrying leg and must fly back to that same location from

---

[1]MDS stands for model/design series (of aircraft).

[2]The mission is not required to end at this time, though, and may be assigned any time in the interval between the Release and Due Dates.

the last cargo-carrying leg. This location is the (air) base associated with the wing to which the mission is assigned. For instance, suppose mission ABC0807P0335, whose cargo itinerary we've listed above, is assigned to the 437th Air Wing – 437AW. The 437AW is located at Charleston Air Force Base, denoted by the location code KCHS. The assigned itinerary for that mission would then be represented as "KCHS RJTY RKSO RJTY PAED KCHS." Note that if that mission were reallocated to a different wing, then KCHS at the beginning and end of the itinerary would be replaced with different location codes.

There is one exception where a mission's itinerary *possibly* may not be expanded when it becomes assigned. This is the case in which a mission's first and last cargo legs begin and end at a location that happens to be the location of an air wing. In this case the scheduling engine can decide to allocate an aircraft from that wing and no positioning or depositioning is necessary. However, it may be necessary (e.g., no aircraft available at that wing) or judicious to allocate an aircraft from a different wing, in which case positioning and depositioning legs would be added.

3.2 Itinerary/Leg Duration

Before a mission is assigned, its duration is unknown other than the fact that all the cargo-carrying legs will take place between the Release Date and Due Date. When an assignment of a unit of aircraft capacity is to be made to a mission, the actual duration of each of its legs can be computed as follows:

1. The unit of aircraft capacity (an airplane) is assigned to the mission from a particular wing if that wing possesses at least one unit of available capacity over the entire duration of the mission.

2. With each wing is associated a particular aircraft type that has a given air speed or velocity.

3. The duration of each leg of the mission is computed as the quotient of the distance between the origin and destination of the leg and the velocity. We will describe how to compute this distance below.

4. The duration of the mission is the sum of the durations of all of its legs (including possibly the positioning and depositioning legs as described above) with no gaps allowed. That is, for the purposes of this model, there is a tight precedence constraint between legs, with the successor beginning immediately (a one-second granularity is assumed) after the predecessor.

For example consider the mission ABC0807P0335 referred to above. This mission requires a C017 aircraft. The 437AW has a C017 wing. The velocity associated with a C017 is 500 knots/hour. The distance of the leg from KCHS to RJTY is 6164 knots and thus its duration is $(6164 \div 500) * 3600 = 44381$ seconds. Similarly we can compute the duration of $RJTY \rightarrow RKSO$ as 4298 seconds, $RKSO \rightarrow RJTY$ as 4298 seconds, $RJTY \rightarrow PAED$ as 21888 seconds, and

$PAED \rightarrow KCHS$ as 22522 seconds, for a total mission duration of 97387 seconds.

There is one exception to this computation of leg duration. That is the case of legs with the same origin and destination as alluded to above. We mentioned the fact that some missions have a simple cargo-carrying itinerary of one leg with the same origin and destination. In the provided data set there are also missions where more than one leg has the same origin and destination. In both cases the duration of that leg may be computed by the function $proportionateDuration$.

proportionateDuration($mission$)
  $duration \leftarrow Touchdown(mission) - Release(mission)$
  $itinLength \leftarrow count(cargoItinerary(mission))$
  Return $round(duration \div itinLength)$

Figure 1: Function proportionateDuration

For example consider the mission 8PH42F801336 whose cargo itinerary is "0312 0312 KLTS 0312 0312 KLTS 313S 313S KLTS 0312 0312." In this itinerary there are three legs $0312 \rightarrow 0312$ and one leg $313S \rightarrow 313S$. Since the Touchdown Date minus the Release Date for the mission is 402120 seconds and there are ten cargo-carrying legs, the duration for all four of these legs is computed the same by *proportionateDuration* as $(402120 \div 10) = 40212$ seconds. All other legs for 8PH42F801336 are computed in the usual manner as described above.

3.3 Aircraft Types

Aircraft (or MDS) Types are specified in the file mds-data.txt. There are only two attributes given for each aircraft type:

- **Name.** This is the unique identifier for the aircraft type, such as "C017" or "KC135." When a mission is scheduled it must be assigned one unit of capacity (an aircraft) from a wing whose Aircraft field matches the Aircraft field of the Mission.

- **Velocity.** This is the average velocity associated with the aircraft type in units of nautical miles per hour. This value is used in computing the duration of time required to fly from one location to another when an aircraft of this type is assigned to a mission.

3.4 Locations

Locations, or ports, are specified in the file port-data.txt. As explained above, a mission comprises a series of legs where each leg constitutes a movement (possibly stationary) from one location to another. Each air wing, as we shall explain shortly, is associated with an air base at a particular location. Attributes for a location are:

- **ID.** This is the unique identifier for the location such as "EDRZ" or "904NW." The former happens to refer to an airport or port and the latter to an air refueling track.[3]

---

[3]Note that some refueling tracks are zero-padded in parts of the input data and not in other parts, and data parsing procedures should take this into account. For instance, the track designator 0312 is equivalent to 312, and 20SW to 020SW.

In the actual domain model an air refueling track is a three dimensional region of airspace in which a refueling (tanker) aircraft refuels a receiver aircraft. For the purposes of the simplified model we will ignore any special significance of a refueling track, and will treat them like any other location on a mission's itinerary.

- **Name.** This is a more descriptive designation of the location ID. For instance, the name associated with location ID "EDRZ" is "ZWEIBRUCKEN" (Germany). This field may be ignored in parsing of the port-data file.

- **Lat(itude).** The latitude of the location in degrees.

- **Long(itude).** The longitude of the location in degrees.

- **Type.** This field takes on the values "PORT" and "TRACK," designating whether the location is a port (airbase) or an air refueling track. For the purposes of this model, this attribute can be ignored.

3.5 Computing distances between locations

To compute the distance between two locations, we make the simplifying assumption that an aircraft flying between those two locations will fly in a shortest-distance "great circle" route; that is a route that conforms to the shortest distance on the curved surface of the earth. This distance may be computed given the (four) inputs of the latitude and longitude of the two locations by the function *gcircle*, below. First we define the following constants:

$pi/180 \leftarrow \pi \div 180.0$
$180/pi \leftarrow 180.0 \div \pi$
$degreeCoeff \leftarrow 90.0 * 60.0$
$auxCoeff \leftarrow 60.0 * 180/pi$

Let *olong* and *olat* be the longitude and latitude of the origin location and *dlong* and *dlat* be the longitude and latitude of the destination location. Then the great circle distance between the origin and destination may be computed by the function *gcircle* as follows:

gcircle($olong, dlong, olat, dlat$)
   $lat \leftarrow pi/180 * olat$
   $latd \leftarrow pi/180 * dlat$
   $m \leftarrow pi/180 * (olong - dlong)$
   $x1 \leftarrow \sin(lat) * \sin(latd)$
   $y1 \leftarrow \cos(lat) * \cos(latd) * \cos(m)$
   Return($degreeCoeff - (auxCoeff * \arcsin(x1 + y1))$)

Figure 2: Function gcircle

Distance computations from the preceding example are summarized in the table 1.

3.6 Air Wings

The remaining element of our model is the air wing. An air wing is a collection of like aircraft at a particular location (base). Wings are specified in the file wing-data.txt, each row in that file indicating available capacity (aircraft) for a given wing and date:

Origin	Destination	Distance (knots)
KCHS	RJTY	6164
RJTY	RKSO	597
RKSO	RJTY	597
RJTY	PAED	3040
PAED	KCHS	3128

Table 1: Sample Distance Calculations

- **Name.** This is the unique identifier for the wing, such as "437AW" or "305AMW."[4] When a mission is scheduled it must be assigned one unit of capacity (an aircraft) from a wing whose Aircraft field matches the Aircraft field of the Mission.

- **Aircraft.** The aircraft type associated with the wing, such as "C017" or "KC010."

- **Base.** The location at which the wing is situated. This will be used to compute the distance of the positioning and depositioning legs once a wing is assigned to a mission. It is assumed that all missions are round trips from and to the base of the assigned wing. As mentioned earlier, if the first and last cargo-carrying legs of a mission begin and end, respectively, at a location that is the base of the assigned wing, then the addition of positioning and depositioning legs is unnecessary.

- **Date.** The day, in MM-DD-YYYY format, for which aircraft capacity is being specified. Note that the first row for each wing has the date 01-01-1993, which is arbitrary, and can be thought of as the "beginning of time."

- **Possessed.** This value represents the number of aircraft the wing possesses. This is not the amount, however, that is available to be allocated on the given date. The reason for this is that on any given day a certain number of aircraft are held in reserve for maintenance and other functions. This value will be used in computations for varying capacity in testing the *TaskSwap* procedure as we will explain shortly.

- **Contracted.** This value represents the number of aircraft the wing has contracted, or made available to the scheduler to allocate on the given date. This number may only be exceeded, or overallocated, by permission of the wing. For the purposes of the simplified AMC problem, we will consider this capacity value to be a hard limit. In addition, for the purposes of this model, we will assume that the last contracted value provided in the data set for each wing will extend out to infinity at that same value (i.e., not drop to zero immediately). For implementation purposes, of course, this capacity value only need be represented for a couple of weeks or so past the point for which actual values are available.

---

[4]Although the name 437AW is unique, the 437AW comprises both a C017 wing and a C141 wing, and these two wings must be considered separately for capacity purposes.

4 Scheduling the Abstract AMC Model

The deployed AMC Allocator application operates in an environment that is ongoing and dynamic. In general, there is no notion of "scheduling from scratch." Rather, a schedule always exists but is continually updated, either due to new missions being added or existing missions being altered. In our research we have simulated that environment to a certain degree, by assuming an existing schedule that has been generated somewhat greedily, and then studied methods for inserting additional tasks into the existing schedule. As our work has progressed, we have discovered techniques that are more applicable to schedule construction – and thus not quite as important to the AMC domain itself – but which might be transferrable to other domains, and which can certainly use the AMC problem as a test-bed.

Later, we will discuss particular requirements for the various experimental results that we publish. At this point we summarize what is required of a scheduling engine to produce a valid schedule given the Abstract AMC problem as we have specified its components above.

- **A feasible assignment.** First of all, an assignment of an air wing to a mission requires making two decisions: which air wing among mission-compatible air wings should be assigned to the mission, and at which time should the mission be assigned along that wing's resource timeline. It goes without saying that the choice of wing and start-time must be both resource feasible and time feasible. I.e., the assigned-start time must be such that all cargo-carrying legs – in tight precedence order – begin on or after the release date of the mission and end on or before the due date. In addition, over the time interval from the beginning of the positioning leg of the mission to the end of the depositioning leg there must be at least one unit of available aircraft capacity for the assigned wing.

- **Priority.** High priority missions take priority in assignment over lower priority missions. This may be encouraged, although not guaranteed, by scheduling missions in priority order. This may not succeed in ensuring that a lower priority mission is able to be assigned when a higher priority one failed since even if a complete scheduling algorithm is employed (which is not advisable given the intractable nature of the problem) it might be the case that at some point when capacity is almost completely consumed, a shorter duration low priority mission could be assigned where a longer duration high priority mission would not be able to be assigned.

  Given the impossibility in proving in the general case that a higher priority mission left unassigned *could* have found a feasible assignment, the strongest statement we can make is that a good schedule is one that minimizes the high priority missions left unassigned after schedule generation and maximizes the total number of missions, irrespective of priority, that are able to be assigned.

- **Cost.** Although there is no strict constraint that the "cost" of a schedule be minimized, this certainly is preferred. For our abstract model we define cost as the sum of mission flight distances (In actuality other factors such as aircraft type are taken into account). Mission distances can be minimized by choosing a wing assignment whose base location minimizes the length of the positioning and depositioning legs. Selecting assignments of minimal cost in this sense has the synergistic effect of allowing more missions to be able to be assigned, since shorter distance missions imply shorter duration missions and thus less resource usage over time.

There are a number of other scheduling preferences that we do not consider for the Abstract AMC problem. One, for example, is to produce resource-balanced schedules. A good schedule is one where capacity usage of aircraft and air crews across wings is similar. It would not be a good policy, then, to assign one wing at 90% of capacity and a wing of like aircraft at 30% of capacity.

5 Experimental Design and Results

Our motivation in writing this paper has been to provide the community with a detailed, though somewhat abstracted, specification of the AMC Problem so that others may reproduce our experimental results, possibly extending them and advancing the state of the art. We assume that those that choose to do so are familiar with the prior research we have published with respect to the *TaskSwap* Procedure [Kramer and Smith, 2003; 2004a; 2004b; 2005] as applied to the AMC domain. For this present effort we have not sought to improve on the results of our prior work, but instead to re-run a subset of the same experiments on the simplified problem and present them as a benchmark. The common objective function in these experiments is to assign as many tasks as possible while respecting task priority, and to do this in an efficient manner.

We document our results for five sets of experiments, all conducted on the same 100 problems.[5] As in our prior work, these 100 problems were generated by assuming that the set of missions to be scheduled (as given in the file mission-data.txt) remains constant, but the capacity of the resources – the air wings (the baseline for which is given in wing-data.txt) – is successively reduced in order to produce harder and harder problems. The 100 problems are divided into five sets of twenty, where the first set is generated by randomly reducing the capacity for each wing between 0 and 10%. The second set is generated by randomly reducing the capacity for each wing between 0 and 20%, and so on to produce five sets of twenty problems. While it is true that this procedure does not guarantee that *all* problems in the second data set, for instance, are harder than all in the first data set, in general the problems for each data set become progressively harder as the likelihood of capacity available to assign missions becomes less.

5.1 Generating Wing Capacity

Recall that the available capacity for aircraft for each air wing is maintained in the file wing-data.txt as two numbers for each

---

wing for a given date: Possessed Capacity and Contracted Capacity. Possessed Capacity does not vary over time. It represents the number of aircraft that the wing "owns." Contracted Capacity, however, represents the number of aircraft that are available to be assigned on a given date, and will vary from day to day depending on the needs of the wing. Those aircraft reserved by the wing to fly training missions or to undergo routine maintenance are subtracted from the Possessed amount to produce the Contracted amount. The Contracted Capacity is the amount the wing has made available to the central schedulers at AMC to allocate to various missions as they see fit.

In order to simplify the process for generating new problems for our experiments, we do not specify a new Contracted Capacity value for each wing for each date, instead we rely on the more compact formulation in the file problem-set.txt. The header of this file is the list of the 16 wings for which data is given in the file wing-data.txt: 437AW-C017, 436AW-C005, 60AMW-C005, 437AW-C141, and so on. Each succeeding row in the file is a vector of 16 numbers, one capacity value for each wing. Each row in the file corresponds to a given problem instance. Problem one, for instance, lists the values 19, 15, 13, 9, 16, 16, 10, 26, 24, 12, 42, 13, 11, 37, 43, 27. For each wing then, we use these values, called $NewPossessed$, to determine from the following formulas how much that wing's Contracted Capacity will be reduced for every day:

For each wing w, the contract reduction CR is
$CR_w \leftarrow (Possessed_w - NewPossessed_w)$
For each wing w and day i the new Contracted Capacity is,
$ContractedCap_{w,i} \leftarrow \max((ContractedCap_{w,i} - CR_w), 0)$
Thus, for the first problem for the 437AW-C017, its contract reduction CR is $21 - 19 = 2$. So for each day its contracted capacity is reduced by 2. Similarly for the 436AW-C005, its contract reduction is $15 - 15 = 0$, so its contracted capacity is not reduced at all.

5.2 Experiment One, Retraction Heuristics

In the first experiment we establish a baseline for the *TaskSwap* procedure on the problem set of problem-set.txt using some of the techniques described in [Kramer and Smith, 2003] and [Kramer and Smith, 2004a]. Recall that *TaskSwap* assumes that there is a schedule in place (possibly generated in a greedy fashion), and that one or more tasks have not been able to be allocated. One by one (in priority order) an attempt is made to schedule these unassignable tasks by temporarily retracting other tasks, and after the target task is assigned re-assigning those retracted. The procedure recurses on those that cannot be re-assigned. This experiment compares four heuristics[Kramer and Smith, 2004a] for selecting which tasks to retract: Min Conflicts, Min Contention, Max Flexibility, and Random. We apply several techniques shown to be effective in [Kramer and Smith, 2004a] for pruning the search space: task pruning, interval pruning, and a depth (recursion) bound of 10. Our results are summarized in Tables 2 and 3.[6]

---

[6]For those interested we can make available complete output data and end schedules for this and all other experiments.

Set	Begin	Rand	MinCont	MinConf	MaxFlex
1	6.4	1.15	1.1	1.65	0.85
2	12.55	5.8	3.75	3.8	3.6
3	27.6	19.95	16.55	17.75	15.15
4	49.8	40.55	36.15	37.15	35
5	100.85	87.95	81.95	83.5	82.55
Avg	39.44	31.08	27.9	28.77	27.43

Table 2: Experiment1: Average Unassignable Tasks

Set	Random	MinCont	MinConf	MaxFlex
1	3.2	4	5.1	1.85
2	13.55	9.1	10.75	5.9
3	49.45	55.35	47.45	24.2
4	125.65	127.9	86.6	62.85
5	373.75	290.55	211	171
Avg	113.12	97.38	72.18	53.16

Table 3: Experiment1: Average Run Time in Seconds

Consistent with our prior work, we see that Max Flexibility on the average outperforms other retraction heuristics both in terms of number of tasks assigned and run time. It does not completely dominate the other heuristics, and in fact for a few problems a policy of random retraction produced the best results. For this experiment "optimal" results (meaning that all tasks were able to be assigned) were achieved using some retraction heuristic for all but 3 of the 20 problems in the first data set. 11 problems in the second data set were solved optimally, and 3 in the third data set.

5.3 Experiment Two, Commitment with Max Availability

In the second experiment we maintain all parameters exactly the same as in the first. We generate an initial schedule in priority order and test the retraction heuristics as before. In this case, though, rather than re-assigning the retracted tasks to their earliest feasible start time, we apply a Max Availability heuristic[Kramer and Smith, 2005] to commit them at the times when all are most likely to be re-assigned. The results are summarized in tables 4 and 5. As can be seen, this results in an improvement across the board in number of tasks assigned. We can also report that 2 more problems in the first problem set are solved to optimality, leaving only one in that set unsolved. Two more problems are solved in the second problem set as well, totalling 13. Averaging over all problem sets, employing the Max Availability heuristic resulted in somewhat slower run times, but looking more closely we see that that is due to much slower times for sets four and five. Runtimes for the first three problem sets actually decreased slightly.

5.4 Experiment Three, Schedule Construction using Max Availability

The third experiment holds constant all of the settings used in experiment two, however in this case we use the Max Availability heuristic to guide the placement of tasks during initial

Set	Begin	Random	MinCont	MinConf	MaxFlex
1	6.4	0.25	0.25	0.15	0.2
2	12.55	2.7	2.15	1.85	1.85
3	27.6	17.35	14.1	13.95	12.9
4	49.8	36.95	32.3	33.75	31.75
5	100.85	83.6	78.05	78.9	77.55
Avg	39.44	28.17	25.37	25.72	24.85

Table 4: Experiment2: Average Unassignable Tasks

Set	Begin	Random	MinCont	MinConf	MaxFlex
1	0.9	0.3	0.1	0	0
2	4.65	2.7	1.6	1.75	1.2
3	18.95	16.3	13.05	12.95	11.35
4	41.15	36.15	31.95	31.9	29.3
5	90.6	81.85	76.75	77.25	75.4
Avg	31.25	27.46	24.69	24.77	23.45

Table 6: Experiment3: Average Unassignable Tasks

Set	Random	MinCont	MinConf	MaxFlex
1	1.4	2.4	2.45	1.3
2	4.35	5.2	6.7	4.4
3	53.2	48.55	38.1	23.45
4	159.1	149.4	112.2	88.8
5	454	326.4	282.95	234.75
Avg	134.41	106.39	88.48	70.54

Table 5: Experiment2: Average Run Time in Seconds

Set	Random	MinCont	MinConf	MaxFlex
1	1	0.75	0.4	0.15
2	7.1	5.05	4.35	2.4
3	53.05	46.2	29.65	20.7
4	136.8	152.25	80.45	52.25
5	363.45	308.9	254.15	206.9
Avg	112.28	102.63	73.8	56.48

Table 7: Experiment3: Average Run Time in Seconds

schedule construction. The results (tables 6 and 7) show that this is a clear improvement [7] over the default greedy schedule construction policy of assignment at earliest feasible start time employed in the first two experiments. Max Availability is used as well during the *TaskSwap* phase, however the results are not as dramatic as during the schedule construction phase. Overall, we end up with more tasks assigned and improved run times. Finally, all 20 problems in the first set are solved to optimality, as are 13 in the second set and 4 in the third problem set.

5.5 Experiment Four, Searching Deeper

The fourth experiment recapitulates the third in all ways except for one parameter change: the depth (recursion) bound is relaxed from a value of 10 to a value of 100. The results (tables 8 and 9) show that this is a fairly expensive trade-off, yielding small gains in tasks assigned but with average run times generally more than twice as long. Optimal solutions are found for three more problems in the second problem set, totalling 16 for that set.

5.6 Experiment Five, Searching Broader

Each experiment we've documented has produced results (in terms of number tasks assigned – actually, number of unassignable tasks remaining) which have improved on the prior experiment. Some of the experiments have done so at the expense of longer run times, but even for the hardest problems, we have shown the *TaskSwap* procedure to be very efficient, in the worst case taking several minutes to terminate. For the fifth experiment we start with the end results of the fourth, and attempt for each problem in our 100-problem set to see if by broadening the search we can possibly improve on the best solutions.[8]

---

[7]Compare the "Begin" columns in tables 4 and 6.

[8]For 40 of the problems this is actually not an issue, as we have already demonstrated that we can quickly reach the optimal solution.

Given the size of the problem space, a complete search is out of the question. Alternatively what we do is employ the "neighborhood search" methods described in [Kramer and Smith, 2004a]. For this experiment we reuse exactly the same parameters as for the prior experiment[9], but test with only the Max Flexibility heuristic for retraction, it having proved on average to be the best. We generate an initial schedule guided by Max Availability and then employ *TaskSwap* as in experiment four. We then conduct ten iterations of search around the Max Flexibility base heuristic, comparing the VBSS technique with the Acceptance Band[10] technique.[Kramer and Smith, 2004a] Both techniques will stochastically pick a task to retract that will typically have a heuristic value similar to that which the Max Flexibility heuristic would select (in some cases selecting the same task), but in some cases VBSS will select a task with a heuristic value far from the "best." Each iteration of neighborhood search begins where the last left off, so if the state has improved, the search does not revert to the initial state.[11]

In table 10 we present a summary of these runs for the five problem sets. Column 1 is the average number of unassignable tasks before the TaskSwap/Neighborhood Search process is initiated. Columns 2 and 3 respectively are the average numbers of end unassignable tasks achieved with 10 iterations of the Acceptance Band and VBSS Neighborhood search process. The final two columns are the average run times for each method. Table 11 lists the final best re-

---

[9]One minor alteration we make is to reduce the depth bound from 100 to 27 as that depth was the greatest successful on the prior run, so searching deeper than that is wasted time. For the neighborhood search phase, however, this depth is doubled to 54, since we search "off" the heuristic somewhat, and can't guarantee that searching deeper won't find a better solution.

[10]Our setting for the Acceptance Band was 0.10.

[11]This semi-exhaustive search process is not for the faint of heart, taking a solid month of cpu time to run the just the 20 problems in problem set five for VBSS and Acceptance Band.

Set	Begin	Random	MinCont	MinConf	MaxFlex
1	0.9	0.2	0.1	0	0
2	4.65	2.9	1.6	1.55	1.1
3	18.95	15.85	12.75	11.9	11.35
4	41.15	35.25	31.55	31.25	29.35
5	90.6	82.8	76.4	76.4	75.15
Avg	31.25	27.4	24.48	24.22	23.39

Table 8: Experiment4: Average Unassignable Tasks

Set	Begin	ABand	VBSS	ABand Time	VBSS Time
1	0.9	0	0	0.3	0.25
2	4.65	0.6	0.4	195.1	159.4
3	18.95	9.5	9.3	6380.85	6257.3
4	41.15	25.25	25.2	26836.55	28386.6
5	90.6	68.9	68.4	65939.2	68777.95
Avg	31.25	20.85	20.66	19870.4	20716.3

Table 10: Experiment5: Neighborhood Search

Set	Random	MinCont	MinConf	MaxFlex
1	0.8	0.75	0.35	0.15
2	14.55	9.9	8.1	5.45
3	145.5	88.5	74.85	50.8
4	392.8	259.05	291.5	148.1
5	927.6	494.4	579.15	459.95
Avg	296.25	170.52	190.79	132.89

Table 9: Experiment4: Average Run Time in Seconds

sult for end unassignable tasks for every problem in the 100-problem set. Note that all problems in the first data set were solved optimally, as were 18 of 20 in data set two, and 5 of 20 in data set three. For those problems for which an optimal solution was not found we present the results as "best known" solutions, which may or may not be further improved upon.

6 Summary

In this paper we have presented a specification of a simplified AMC Allocator domain, so that others may implement it as a test bed for multi-mission multi-capacity scheduling problems. In particular we reproduce our prior work with the *TaskSwap* procedure in a series of experiments that will allow extension of and comparison to that research.

Acknowledgements

Problem	Set1	Set2	Set3	Set4	Set5
1	0	0	0	29	30
2	0	0	7	49	51
3	0	7	20	85	7
4	0	0	1	32	11
5	0	0	0	7	125
6	0	0	16	6	25
7	0	1	19	21	35
8	0	0	10	11	92
9	0	0	8	9	19
10	0	0	24	4	194
11	0	0	0	53	130
12	0	0	2	4	50
13	0	0	19	12	60
14	0	0	0	1	56
15	0	0	27	18	38
16	0	0	2	36	68
17	0	0	7	25	93
18	0	0	0	8	64
19	0	0	20	49	98
20	0	0	1	32	118

Table 11: Experiment5: Final Unassignable Tasks

References

[Becker and Smith, 2000] M.A Becker and S.F Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proc. 5th Int. Conf. on AI Planning and Scheduling*, pages 32–41, Breckenridge CO, April 2000.

[Kramer and Smith, 2003] L.A. Kramer and S.F. Smith. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proceedings 18th International Joint Conference on Artificial Intelligence*, Acapulco Mexico, August 2003.

[Kramer and Smith, 2004a] L. A. Kramer and S. F. Smith. Task swapping for schedule improvement, a broader analysis. In *Proc. 14th Int'l Conference on Automated Planning and Scheduling (ICAPS-04)*, Whistler BC, June 2004.

[Kramer and Smith, 2004b] L. A. Kramer and S. F. Smith. Task swapping: Making space in schedules in space. In *Proc. Fourth International Workshop on Planning and Scheduling for Space (IWPSS-04)*, Darmstadt Germany, June 2004. European Space Agency.

[Kramer and Smith, 2005] L. A. Kramer and S. F. Smith. Maximizing availability: A commitment heuristic for oversubscribed scheduling problems. In *Proc. 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, Monterey CA, June 2005.

[Smith *et al.*, 2004] S. F. Smith, M. B. Becker, and L. A. Kramer. Continuous management of airlift and tanker resources: A constraint-based approach. *Mathematical and Computer Modeling – Special Issue on Defense Transportation: Algorithms, Models and Applications for the 21st Century*, 39(6-8):581–598, 2004.